Naval Research Laboratory

Washington, DC 20375-5320

NRL/FR/5510--95-9766

A Paradigm to Assess and Evaluate Tools to Support the Software Development Process

JAMES A. BALLAS
JANET L. STROUP

Navy Center for Applied Research in Artificial Intelligence
Information Technology Division

May 17, 1995

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 17, 1995	Final Report

4. TITLE AND SUBTITLE

A Paradigm to Assess and Evaluate Tools to Support the Software Development Process

5. FUNDING NUMBERS

PE - 61153N
TA - RR0150801

6. AUTHOR(S)

James A. Ballas and Janet L. Stroup

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/FR/5510--95-9766

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Officer in Charge
Naval Surface Warfare Center
Dahlgren Division Detachment, White Oak
Silver Spring, MD 20903-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

The purpose of this research is to develop and evaluate software prototyping tools that are used to produce "an executable unit that demonstrates particular aspects of the behavior or functionality of the desired software product." It was conducted with a particular perspective of assuming how the tools support an aspect of software design that has received little attention: exploration of design space. To pursue this perspective, an initial definition of design space exploration was developed and hypotheses were proposed on what outcomes would be observed if a tool supported design space exploration. Finally, techniques were designed to obtain data to test the hypotheses in a general manner. Data were collected with these techniques during a session in which a particular tool was used to design software. The results provided some support for the hypotheses and suggested options for further refinement of the methodology. The transcribed observational data supported post-hoc analysis that revealed aspects of the software development that occurred in the two-day session.

14. SUBJECT TERMS

Software development process
Software prototyping tools
Software design
Design space exploration

15. NUMBER OF PAGES

42

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18
298-102

CONTENTS

A PARADIGM TO ASSESS AND EVALUATE TOOLS TO SUPPORT THE SOFTWARE DEVELOPMENT PROCESS

INTRODUCTION

This study was part of a larger project called the "ProtoTech HiPer-D (High Performance Distributed Computing Program[1]) Joint Prototyping Demonstration Project." The Prototyping Technology (ProtoTech) program is a research effort supported by the Advanced Research Projects Agency (ARPA) to develop advanced prototyping languages. The HiPer-D Demonstration was a joint effort of the ProtoTech community and the Office of Naval Research to explore the applicability of prototyping technologies to realistic military problems. An important first step in the HiPer-D project was to identify an appropriate problem for study, one of suitable complexity to demonstrate the technology realistically, yet small enough to be addressed by a limited number of people in a limited period of time. The geometric region (GEO) server problem met these criteria; it was also of particular interest to the Navy for its relevancy to the planned AEGIS weapons system upgrade. The capability to assign trackable objects to their correct geometric regions in space is an essential function in the AEGIS system and currently spread across multiple algorithms. Consolidation into a single GEO server has the potential for significant system improvement. The HiPer-D Demonstration comprised several independent teams of prototyping experts, each addressing the GEO server problem in a different prototyping environment. In addition, several related efforts were undertaken to examine specific aspects of the problem.

Our work was conducted to assess a paradigm for evaluating how well a prototyping tool supports software development. Prototyping tools are used to produce "an executable unit that demonstrates particular aspects of the behavior or functionality of the desired software product" (Lee et al. 1994). This assessment included data collected at a two-day interactive software development exercise at the Naval Surface Warfare Center, Dahlgren Division (NSWCDD) involving a prototyping tool expert from Kestrel Institute and two domain specialists from NSWCDD using the Kestrel Interactive Development System (KIDS). KIDS was selected as the software tool for this exercise for several reasons. It is a product that is relatively mature and has been used in a variety of domains. It also is designed to support a particular software development process. This enabled us to make predictions about the types of behavior we would expect and thereby assess not only the tool but perform a meta assessment on the evaluation methodology. Finally, the tool supports a software development process that is abstract and closely coupled to the theory of the domain. Paradigms to evaluate software development at an abstract level are not available. The focus at this level is the representation of the domain in software, and actions taken to develop and change this domain. This focus is consistent with the general hypothesis that was explored. This exercise provided us an opportunity to further our research in the exploration of design space.

---

1. This is a project to develop a new architecture for the next generation of the AEGIS Weapons System. It is to be jointly developed by three organizations: Johns Hopkins University/Applied Physics Laboratory (JHU/APL), General Electric (GE), and the Naval Surface Warfare Center at Dahlgren, Virginia (NSWCDD) supported by the Computer Sciences Corporation (CSC). The work was divided into three thrusts: 1) to evaluate a wide range of architectures and combat system issues and choose a recommended architecture; 2) to evaluate promising technologies provided by the Advanced Research Projects Agency (ARPA); and 3) to evaluate and extend emerging HiPer-D technologies, methods, and tools. The technologies provided by ARPA under Thrust 2 include the ISIS distributed computing workbench from Cornell University, INTEL's Paragon supercomputer, and the MACH microkernel developed by Carnegie Mellon University.

Manuscript approved Oct. 20, 1994.

EXPLORATION OF THE SOFTWARE DESIGN SPACE

The general hypothesis being tested is whether a tool supports exploration of the design space. The evaluation of this hypothesis implies an understanding of what design space exploration means. If one considers design space exploration as one part of the software engineering process, then initial definitions of this concept can be derived from a framework of design methodologies developed by Song and Osterweil (1992). Their framework is a method-component hierarchy and includes four abstract types: Concept, Artifact, Representation, and Action. Using their notions of these types, the following preliminary definitions were developed.

Definitions

Design space exploration consists of actions that produce different representations.

Representations are descriptions or specifications of design artifacts.

Actions are physical and/or mental processing steps used to produce or modify an artifact.

These preliminary definitions need to be tailored somewhat for the particular tool that is evaluated. Accordingly, examples of representations and actions that are possible within the KIDS tool were generated to guide the development of the data collection methodology. The representations created in KIDS are the domain theory, specifications, and programs described in the Refine language (upon which KIDS is built), and the Lisp code. Examples of actions within the KIDS tool that produce different representations are:

Develop theory: produce the initial representation
Return to initial/previous/next state: reset representation to another state
Focus (i.e., select subset of specification): select a subset of the representation for subsequent actions
Apply transformation
Select inference mode
Fold, unfold a specification
Simplify an expression
Select algorithm tactics
Select compiler

A record of the actions taken while solving the GEO server problem was made by observation and audio recording of the participants' statements, and by periodic storage of the solution file. These data were examined to assess whether the tool supports design space exploration. In order to develop some detail about the expected effects of the tool on exploration, specific hypotheses were generated before the data collection session. These specific hypotheses provided further guidance about the type of data that should be collected.

Specific Hypotheses

The following section presents specific hypotheses (Ho1 through Ho5), followed by data that were collected and analyzed to test the hypotheses. At this point in the development of an evaluation paradigm, the hypotheses can not be tested statistically.

Ho1: Alternative representations can be generated and actions taken on them.

Distinct, different alternatives will be represented and actions taken on them. The data to evaluate this hypothesis will consist of identifying alternative representations that are generated during the challenge problem design session. The issue here is how to distinguish between alternatives. In many

cases, this distinction could come from the alternative menu options that are offered to the user, such as selecting a particular algorithm tactic or selecting a particular compiler. This choice will produce a different specification. Another difference in the representation might be the feedback to the user that occurs when the inference mechanism is operating. The user has several options that will provide different types of feedback and different opportunities for intervention.

Ho2: Design exploration will involve evaluation and comparison of alternatives.

Evaluation actions will be taken on alternatives followed by modification or generation of an alternative. This hypothesis further specifies what exploration of alternatives means by focusing on particular actions—those which produce an evaluation of the artifact. Data for this hypothesis could come from the final stage of development, when the specification is compiled and executed. At that point, we may observe the KIDS users examining software metrics such as size and speed. However, during the development process, there may be verbal comments alluding to comparisons of alternative designs. One of the windowing configuration options available in KIDS places two program viewing windows side by side. Use of this configuration would be evidence for this specific hypothesis.

Ho3: Alternative representations may support evaluation and comparison of design alternatives.

This hypothesis refers to the possibility that the type of representation may or may not support comparison of alternatives. For example, an option in KIDS is to fold/unfold the specification, producing less and greater detail respectively, and allowing the user to compare alternatives at different levels of detail. Evidence for this hypothesis will be actions taken to change the representation form, which do not really add or subtract from the complete specification.

Ho4: Design space exploration will consist of a fan-out generation and subsequent culling of artifacts.

A history trace of alternatives generated will show an increase in the number of alternatives, followed by a narrowing of the alternatives toward a particular solution. This hypothesis is suggested since it is an intuitive model of the creative process.

Ho5: Design exploration will involve an examination of prior actions and representations.

Evidence of the user's examination and analysis of prior actions and alternatives will occur in two ways using the KIDS tool. The first will be through conversations in which the users will reflect upon and comment upon prior actions. The second will be through viewing the Output History Pane and taking actions on prior derivations listed within this window.

Hypothesis 5 is a simple statement of a process that is complex and little understood. It refers to a meta level of software development in which a person is evaluating what has been done. This examination can be observed in some ways. But the examination is being done to support an understanding of what has been done and the planning for further actions and development of representations.

Elaboration of this hypothesis can take on different forms depending on whether design space exploration is conceived of as problem solving, creative construction, invention, etc. This point returns us to the particular perspective of this research: to develop and evaluate a paradigm to monitor design space exploration. Once we have the capability to follow a process of exploring the design space, we will be able to be more specific about what it actually is.

USABILITY EVALUATION

In addition to developing a technique to evaluate how the tool would support design space exploration, there was interest in evaluating the usability of the tool. The evaluation of this aspect of

software development used traditional usability procedures. These procedures were partially tailored for the KIDS tool. For example, one of the evaluation criteria is user performance on a benchmark task. With KIDS, the primary user tasks are those in the software process model on which KIDS is based. These tasks, and the subtasks within each of them, are as follows:

Specify domain theory:
- Specify functional constraints on input/output behavior
- Generate abstract expression
- Create rules
- Derive laws

Convert specification to code:
- Design algorithm
- Simplify
- Partially evaluate
- Refine data types

The performance of these tasks was assessed by real-time observation and inquiry, and by a post-session questionnaire. Real-time observation was performed using the annotation sheet shown in Fig. 1. Four columns were used to record the time of the observation, the current user task, the program function or window in which the user's actions were located, and the details of the observation. Observations were detailed so that the following types of information would be obtained for later analysis:

- Task completion time and errors
- Number of options/alternatives explored
- Critical incidents (both positive and negative)
  - Inability to find/open/close/save "work"
  - Recognition of success
  - Recognition of other programming options
  - Evaluation of algorithm performance vis-a-vis different implementations
  - Recognition of and recovery from error
  - Error-free performance on first attempt

Time	Task	Location	Annotation

Fig. 1 — Annotation sheet for real-time observations

In addition to observations, occasional inquiries were made to clarify comments and activities. In particular, clarification was sought on issues related to the form of the representation that was being developed (e.g., why did you decide to choose this form of representing the domain?). In order to have a record to clarify the observations and inquiries, a continuous audio recording was made of the sessions.

Finally, a post-session questionnaire was developed that included open-ended questions about the strengths and weaknesses of the tool and solicited recommendations for improvement. A series of these

questions was formatted for the particular design model in KIDS. These questions asked how the tool supported the specific phases in the design process and what improvements would be recommended to support these phases. This tailoring is consistent with the IEEE standard to evaluate Computer-Aided Software Engineering (CASE) tools as described below. The questionnaire also included a table of design heuristics that have been used in an approach called heuristic evaluation to find usability problems in interfaces (Nielsen 1992). Heuristic evaluation uses a small set of individuals to evaluate an interface according to design principles.

COMPARISON TO IEEE STD 1209-1992

IEEE STD 1209-1992 (IEEE 1992) outlines a recommended practice to evaluate and select CASE tools. Although the tools of interest here might not be considered CASE tools in some respects, it is useful to compare our approach with this standard of recommended practice. The standard outlines several evaluation process models. The one most comparable to our approach is the "evaluation for future reference" model. In this model, the tool is being evaluated on all relevant criteria and the results are made available for future reference. In the other models, a tool selection phase is involved that introduces issues that are not present when a single tool is being evaluated (e.g., weighting of selection criteria). Two steps are part of this model: the development of tailored criteria and the evaluation itself. Criteria tailoring is the selection and definition of a set of criteria whereby the characteristics of the tool are quantified and measured. The types of criteria listed in the standard include reliability, usability, efficiency, functionality, maintainability, profitability, and a general category. Many of these criteria are similar to those used in the ProtoTech challenge problem paradigm. Two of them, usability and functionality, were the key criteria presented in the initial briefing of this proposed paradigm to the ProtoTech community. Details on how to quantify these two criteria were developed prior to the data collection session. These details reflected some tailoring for the particular tool. Specifically, examples of the types of actions that could be taken in the KIDS tool were listed as performance indicators to test the hypotheses. Thus, the tailoring recommended in the IEEE standard was partly completed. As described earlier, usability was assessed through observation and inquiry and through completion of a post-session questionnaire.

In the present effort, there is particular interest in investigating how to evaluate exploration of the software design space. Although the IEEE standard does not explicitly address this aspect of functionality, it does include some specifics that would seem to be related to design space exploration. It decomposes specific functionality criteria into three classes: those related to the operating environment, those related to particular life-cycle phases, and those common across all life-cycle phases. The life-cycle phases are decomposed into modeling, implementation, and testing, and it is the criteria listed under modeling that may be related to design exploration. Modeling criteria are intended to assess a tool's capability to support the identification of software requirements, to transform requirements into design, and "*to express software design*" (italics added for emphasis). Specific modeling criteria include: diagramming, graphic analysis, requirement specification entry and editing, requirement specification language, design specification entry and editing, design specification language, data modeling, process modeling, simulation, prototyping, screen generation, traceability, specification consistency and completeness checking, other analyses, and report writing. Thus the standard supports the waterfall software development model and places emphasis on the capability to transform requirements into design specifications. However, it is useful to note the importance the standard places on the capability of a tool to support the expression of the software design. Clearly, design space exploration requires and will be enhanced with adequate design expression tools. The standard also incorporates these specific criteria mechanisms to evaluate the design such as simulation, data and process modeling, and traceability. It might be a useful exercise to tie some of these specific criteria into a prototypical process model for design space exploration showing the feedback loops and iteration that would be expected and theoretically should be supported to enhance the exploration of the design space.

RESULTS

Prior Hypotheses

The data for these results come from four sources: the real-time notes (a transcribed and edited version of these notes is attached as Appendix A), the audio tapes (used to edit and refine the real-time observations), the solution files that were saved periodically (the last solution file is included in Appendix B), and the post-session questionnaire that was completed by two persons from NSWCDD (Appendix C). Because an objective of this research is to refine a methodology to assess the functionality and usability of prototyping tools, the results present two perspectives: a description of the type of assessment available in the employed form of the techniques, and an assessment of the techniques themselves. The first perspective on the results is presented within the context of the detailed hypotheses that were made.

Ho1: Alternative representations can be generated and actions taken on them.

An example of alternative representations is the initial and alternative versions of the function HIPER-D which places contacts (objects) into the particular zones (see Fig. 2). The initial version simply specifies the mapping function. The alternative version divides this mapping function into three cases: the empty set of zones, the singleton set of zones, and the remaining cases. The alternative version was produced by applying the divide and conquer tactic to the initial function, followed by the application of simplification tactics. The divide and conquer tactic generated an alternative software design, upon which further actions were taken. Therefore at least one data point supports this hypothesis.

Initial version

```
function HIPER-D
  ( fo : FLATLAND | ... )
  returns ( contact-map : map(CONTACT, set(ZONE))
| contact-map = {| c -> { z | (z:ZONE) z in all-zones(fo) & in-zone(c,z)}
|(c:CONTACT) c in contacts(fo) |})
```

Alternative version

```
function HIPER-D-2 (CNTKS-7: set(CONTACT), ZNS-9: set(ZONE))
 returns
  (Z-192: map(CONTACT, set(ZONE))
    | Z-192
       = {| C -> {Z | (Z: ZONE)
                Z in ZNS-9 & IN-ZONE(C, Z)}
          | (C: CONTACT) C in CNTKS-7 |})
 = if CNTKS-7 = { } then {| |}
   elseif CNTKS-7 less! arb(CNTKS-7) = { }
       then {| C -> {Z | (Z: ZONE)
                  Z in ZNS-9 & IN-ZONE(C, Z)}
             | (C: CONTACT) C in CNTKS-7 |}
   else let (Y-OP-3
           : tuple
             (set(CONTACT), set(ZONE), set(CONTACT), set(ZONE))
           = HIPER-D-2-DECOMPOSE-USING-UNION-DESTRUCTOR
             (CNTKS-7, ZNS-9))
        HIPER-D-2(Y-OP-3.1, Y-OP-3.2)
         +* HIPER-D-2(Y-OP-3.3, Y-OP-3.4)
```

Fig. 2 — Initial and final versions of function HIPER-D

Ho2: Design exploration will involve evaluation and comparison of alternatives.

Three types of data were suggested for evaluating this hypothesis. The first was that the alternatives would be coded and executed and comparisons made using standard software metrics. This outcome was not observed during the session since a fully executable solution was not completed in the time set aside. However, given further time it is expected that this type of data would be observed.

The second type of data was verbal comments made during the development process. There were several examples of this. Early discussion focused on defining the objects in the domain, especially the types of contacts and the types of zones. The initial list of zones came from the problem description and included four types: WEAPON, ENGAGEABILITY, SLAVE-DOCTRINE, and TIGHT. These were abstracted to two types, WEDGED-ANNULUS and POLYGON, which were each to be defined in a manner that would cover virtually all cases. The properties of the WEDGED-ANNULUS were defined so that WEAPON, ENGAGEABILITY, and SLAVE-DOCTRINE zones could be handled by this object. The properties of POLYGON were defined as the position of a sequential list of vertices that would support the inclusion of both convex and concave polygons and any number of vertices. The decision to opt for generality was made in the specification phase of design and was made with some verbal consideration of the implications. Later, when it came to specifying whether a contact location was within a polygon in the function IN-POLYGON, a particular algorithm had to be implemented and the coding implications of this completely generalized alternative became apparent. Much of the afternoon of the second day was spent generating code for an IN-POLYGON function. This coding would not have been necessary if a library function were readily available. In fact, there is a C function that checks whether a point is within a polygon. The ability to incorporate this function as a call would have enhanced the development of the generalized solution sought in this session and perhaps supported the further evaluation and comparison of generalized designs to designs that were more specialized. In this instance, design space exploration would have been enhanced if this particular function could have been found through a library search and once found, quickly incorporated into the solution.

The third type of data for evaluating this hypothesis was a side-by-side windowing configuration. There were at least two instances of this. The first was about 17:57 on the first day (see Appendix A). The tutor opened two editing buffers to enter constraints on recently entered types. These two buffers were used to maintain consistency between an initial specification and an elaboration of this specification. The second instance was 16:16 on the second day when two buffers were again used. In this instance, the buffers were used to implement and test the code. In this instance, additional functions were created that had to be consistent with representations created earlier. In both cases, the representations being compared are not alternatives in the strict sense but are different representations of the domain, which have to include some elements that are consistent. These instances were captured by an observer query and observation.

Ho3: Alternative representations may support evaluation and comparison of design alternatives.

The possibility in this hypothesis is that evaluation and comparison of alternative designs may be better supported by some forms of representation. The a priori example was the folding or unfolding of a specification. This was not observed in the session. However, following up on the discussion of the previous hypothesis, the implications of the generalized representation of polygon became more evident as the particular algorithm to implement IN-POLYGON was written. In other words, the difficulty of implementing the generalization representation may not have been as apparent when the zone class representation was decided.

Ho4: Design space exploration will consist of a fan-out generation and subsequent culling of artifacts.

There was one instance that supported this hypothesis. When the function HIPER-D was expanded with the Divide and Conquer tactic, redundant code was generated for the empty set case. The subsequent simplification for this case reduced the artifact. It was expected that this hypothesis would also be revealed in the generation of different design representations, not just detailed specifications. It was also expected that artifacts at a higher level would be generated, then pruned as the alternatives were evaluated and compared. Instances of this were not observed in the artifacts, although examples of this process are suggested by the audio record. In particular, during the process to define whether a point was within a polygon, several alternative algorithms were generated and discussed. One was finally settled upon.

Ho5: Design exploration will involve an examination of prior actions and representations.

There were several instances that are consistent with this hypothesis. Two that may be particularly relevant were instances when a higher level object description was modified after and during the process of specifying details at a lower level. For example, in the process of defining the function IN-WEDGED-ANNULUS, the insight occurred that the whole zone did not have to be passed, just the zone shape. This required a rewriting of the function IN-ZONE, which had already been generated.

Supplementary Analyses

Observational Data

The combination of real-time observation with audio recording produced a substantial data record that might provide the basis for extensive analyses. Some examples of possible analyses follow.

Observations were coded and categorized as follows and can be used for further analyses:

K: Key incidents
O: Observations
Q: Observer queries
R: Responses to queries
T: Tutor comments or queries
U: User comments or queries

A distribution of these codes by type is shown in Table 1. Fifty-eight of the 246 codes were incidents of key interest. Most of the key incidents (29) were errors (see Table 2). Very few were representation decisions, a key behavior for the assessment of the a priori hypotheses. The low number means that either very few decisions were being made, or that the decisions were not captured with this observation procedure. It is apparent that further refinement of a method to trace the development and modification of the domain representation would be useful.

A second analysis of the observation data focused on task timelines and task context. The intent was to determine the contextual shifts that occur in the software development process. Studies of user interaction with intelligent systems have pointed to the disruptive effects of contextual shifts (Malin et al. 1991). An example is when an operator of a fault diagnosis system must mentally shift from thinking about the domain to thinking about how to use a computer interface to control the system or obtain information about system status. This disruptive effect of the interface is called "wading through the interface." Similar interference may occur in software development with advanced prototyping tools. The development process might be interrupted by a computer interface and development environment that imposes contextual switching As a simple example, if in the process of coding an algorithm, users have to search for the name of a library, they may forget some of the subsequent logic that had been mentally

sketched, especially if the operations imposed by the interface require mental work and activity that is irrelevant to specifics of the domain.

Table 1 — Frequencies of Different Types of Observations

Type of Observation Code	Frequency
Key Incident	58
Observation	92
Observer Query	10
Response to Query	20
Tutor Comment/query	38
User Comment/query	28

Table 2 — Classification of Key Incidents Observed

Type of Key Incident	Frequency
Error	29
Rewrite/Revision	5
Representation Decision	9
Misc.	15

To assess contextual switching, the task and task times were imported into commercial project management software to generate graphical timelines. Additionally, the task context was coded into one of the following categories:

A) Demonstrate tool
B) Understand problem
C) Define theoretical objects
D) Define attributes of objects
E) Define functions
F) Refine and compile specification
G) Perform save/load/find operations

Figure 3 shows the timeline. The tasks are sorted first by task context and second by starting time. Note that activity on the first day follows a waterfall pattern. There is some contextual switching between categories C and D. It's unlikely that this switching would be confusing except perhaps on the switches from D back to C. Note that this pattern was iterated. Activity on the second day shifts between contexts to a greater degree, and the switching is less systematic. This shifting is due to several factors. On the second day, the solution was becoming complete, which meant that the files were being saved more frequently and later phases of the development process (i.e., compile) were possible. Secondly, in addressing a difficult aspect of the challenge, the users were searching and exploring options and moving out of a systematic development pattern. It would be particularly interesting to discover the development processes that occur when people address the more difficult aspects of the problem. Finally, as the solution evolved, refinement and compilation were possible, which in turn led to examination and revision of earlier products.

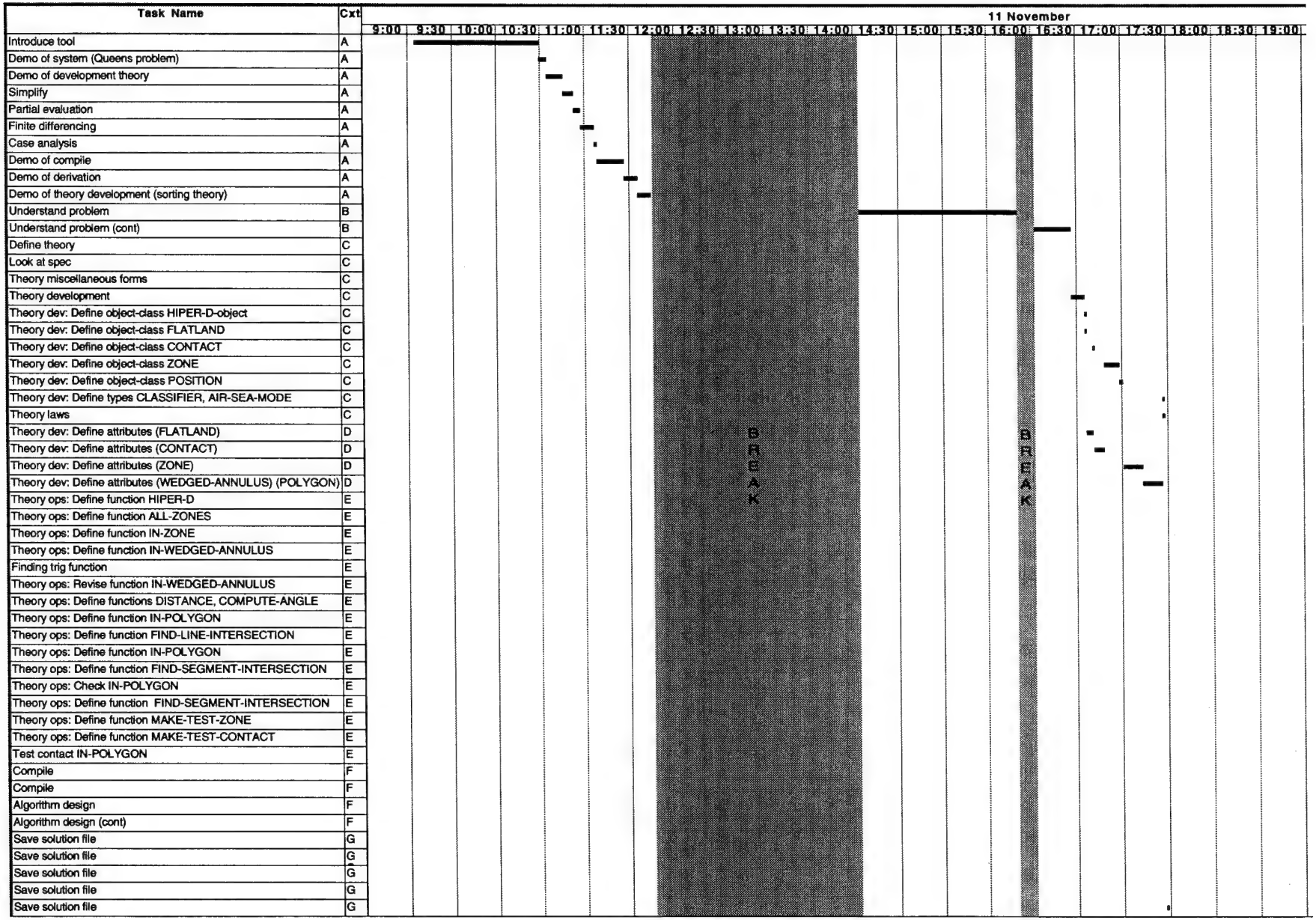


Fig. 3—Task timeline sorted by task context and starting times

Task Name	Cxt
Introduce tool	A
Demo of system (Queens problem)	A
Demo of development theory	A
Simplify	A
Partial evaluation	A
Finite differencing	A
Case analysis	A
Demo of compile	A
Demo of derivation	A
Demo of theory development (sorting theory)	A
Understand problem	B
Understand problem (cont)	B
Theory development	C
Theory dev: Define object-class HIPER-D-object	C
Theory dev: Define object-class FLATLAND	C
Theory dev: Define object-class CONTACT	C
Theory dev: Define object-class ZONE	C
Theory dev: Define object-class POSITION	C
Theory dev: Define types CLASSIFIER, AIR-SEA-MODE	C
Theory laws	C
Define theory	C
Look at spec	C
Theory miscellaneous forms	C
Theory dev: Define attributes (FLATLAND)	D
Theory dev: Define attributes (CONTACT)	D
Theory dev: Define attributes (ZONE)	D
Theory dev: Define attributes (WEDGED-ANNULUS) (POLYGON)	D
Theory ops: Define function HIPER-D	E
Theory ops: Define function ALL-ZONES	E
Theory ops: Define function IN-ZONE	E
Theory ops: Define function IN-WEDGED-ANNULUS	E
Finding trig function	E
Theory ops: Revise function IN-WEDGED-ANNULUS	E
Theory ops: Define functions DISTANCE, COMPUTE-ANGLE	E
Theory ops: Define function IN-POLYGON	E
Theory ops: Define function FIND-LINE-INTERSECTION	E
Theory ops: Define function IN-POLYGON	E
Theory ops: Define function FIND-SEGMENT-INTERSECTION	E
Theory ops: Check IN-POLYGON	E
Theory ops: Define function FIND-SEGMENT-INTERSECTION	E
Theory ops: Define function MAKE-TEST-ZONE	E
Theory ops: Define function MAKE-TEST-CONTACT	E
Test contact IN-POLYGON	E
Compile	F
Compile	F
Algorithm design	F
Algorithm design (cont)	F
Save solution file	G
Save solution file	G
Save solution file	G
Save solution file	G
Save solution file	G

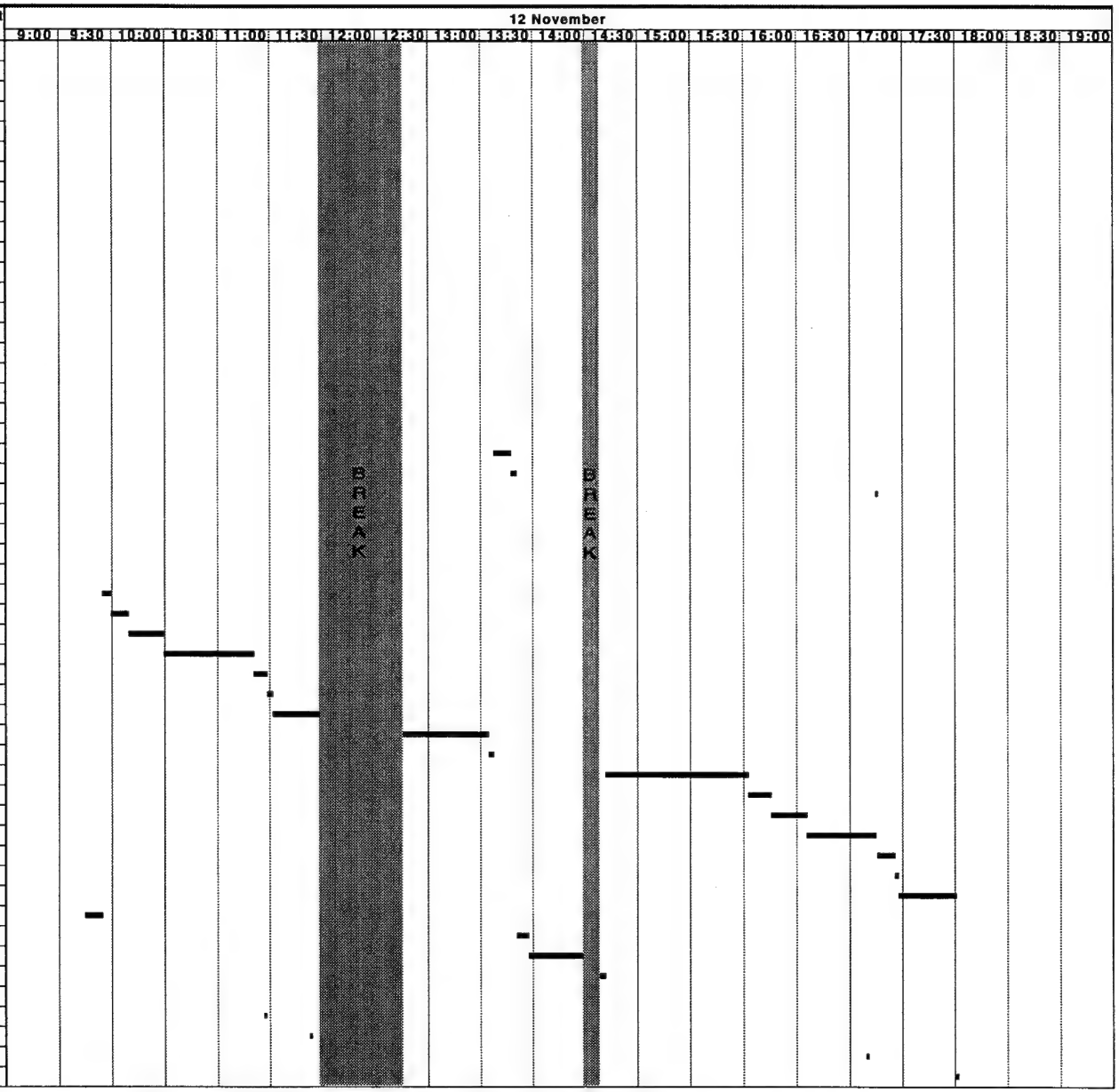


Fig. 3 (continued)—Task timeline sorted by task context and starting times

Questionnaire

Appendix C presents the questionnaire. Because only two users were involved in this study, the data collected from the questionnaire are sparse. Furthermore, the users had only a viewgraph presentation and brief demonstration of the tool during the morning of the first day—much less than the standard 5-day training normally given to understand the tool. However, even with only a sample of two novice users, the results of the questionnaire do provide information about suggested changes to the interface that may be useful for the developer.

Methodology Evaluation

The data collected can provide instances that support the hypotheses, but are inappropriate to use for statistical tests of any of the hypotheses. For further development of the evaluation methodology, alternative hypotheses will have to be generated a priori and the data collection technique refined. Specifically, prior clarification or definition of instances that support competing alternatives will have to be made to avoid confirmatory biasing effects.

Observation Data Capture and Analyses

As illustrated in the supplemental analyses, the transcribed observational data can support some interesting "discovery" analyses. These analyses will require an accurate and consistent method of recording the data. While much of the data can be obtained from an audio record, and more from a video record (which was not done in this instance), a real-time recording methodology is important in order to capture the key incidents. The real-time record can also focus any subsequent transcription of the tapes.

Additional analyses could be performed with an improved data capture procedure. For example, analysis of errors could derive the causes of the error. But this would required detailed data on the error action and its context, information that could come from a video record and behavioral actions.

Two caveats should be noted about the observational data collected. First, the amount of data collected was due in part to the particular composition of the design team: a tool expert and two domain users. The tool expert was only generally familiar with the domain; the users had little knowledge of the tool. This composition facilitated the generation of a verbal record, probably more so than if a single person was involved and asked to think aloud or record their process. A major limitation of this composition is that the tool expert had two roles to assume: a tutor for the tool and a software designer. This duality would be mixed in the data record, especially the verbal record and perhaps in the artifact record. Such a mixture would confound two phases in the evaluation of a tool: its learnability and its effectiveness in developing a design solution.

Questionnaire

The questionnaire did not include a rating scale, but User1 added one, after consulting with the experimenter. Usually this would be highly inappropriate, but in this case, the users were instructed to recommend changes to the questionnaire instrument itself. User1 employed the response scale in answering each of the questions that solicited an evaluation of a particular function of the tool. User1 also provided verbal labels for the units of the response scale and in this sense, the numeric scale was simply a shorthand notation for the verbal judgments. Rather than presenting an extended discussion about the validity and merit of a numeric response scale, a few comments will be made. First, although the user felt comfortable with using a response scale, the meaning of the scale units (e.g., excellent, good, above average,...) is uncertain in the absence of a standard or a set of definitions. Furthermore, in the absence of a response scale, the respondent may be inclined to provide greater detail. For example, in response to question 9 ("How well does the tool support exploration of the design space?"), User2's response that the

tool provides "an opportunity but doesn't force it or explicitly encourage it" gives some information about how this user views the design capability in the tool.

It is apparent in both users' responses that definitions of some concepts must be added. This refinement of a questionnaire comes about through the normal process of pilot testing. One feature of the questionnaire that seemed to solicit information in a useful manner was to direct a set of three questions toward each of the particular functions of the KIDS tool. The first question in the set asks about the general value of the tool for the function (e.g., Q5 "How well does the tool support algorithm design?") and the second and third questions solicit details about the best features for this function and recommended changes or additions. Sometimes the answers will be contradictory. For example, in response to question 5, User1 indicated that one of the best features was the algorithm design library while User2 recommended some library-like features.

Finally, it is also apparent that clarification of many of the users' responses would be helpful. This suggests that a structured interview would be more appropriate than a self-administered questionnaire. Because the user community for these advanced prototyping tools will be relatively small, a structured interview is feasible from a sampling perspective. And since many of the questions will address aspects specific to a particular tool, prior refinement of a questionnaire will be impractical. Thus it is recommended that structured post-session interviews be used rather than self-administered questionnaires.

ACKNOWLEDGMENTS

We benefited greatly from the advice and consultation of Debbie Hix and Lisa Achille on design and methodology, and Ralph Wachter on objectives and strategy. Harry Crisp contributed management and logistic support. Doug Smith consented to using KIDS as the test case and served as the expert tool user. Chang Nguyen and Mark Wilson served as the domain experts. We received valuable review comments from Lisa Achille, Helen Gigley, Debbie Hix, Rob Jacob, and Doug Smith. This work was sponsored by the Office of Naval Research through the Naval Surface Warfare Center, Dahlgren, VA.

REFERENCES

IEEE Std 1209-1992 (1992). IEEE Recommended practice for the evaluation and selection of CASE tools. IEEE: Piscataway, NJ.

Lee, J.A.N., B. Blum, P. Kanellakis, H. Crisp, and J. A. Caruso (1994). ProtoTech HiPer-D Joint Prototyping Demonstration Project. Final Report Version 1.0, Naval Surface Warfare Center Dahlgren Division: Dahlgren, VA.

Malin, J. T., D. L. Schreckenghost, D. D. Woods, S. S. Potter, L. Johannesen, M. Holloway, and K. D. Forbus (1991). Making Intelligent Systems Team Players: Case Studies and Design Issues. NASA Technical Memorandum 104738, Lyndon B. Johnson Space Center, NASA: Houston, TX.

Nielsen, J. (1992). Finding usability problems through heuristic evaluation. In proceedings of CHI, 1992 (Monterey, California, May 3-May 7, 1992). ACM, New York, 1992, pp. 373-380.

Smith, D. R. (1990). KIDS: A Semi-Automated Program Development System, *IEEE Trans. Software Engin.* **16**(9), Special Issue on Formal Methods, Sept. 1990, pp. 1024-1043.

Song, X. and L. J. Osterweil (1992). Toward objective, systematic design-method comparisons. *IEEE Software*, May, pp. 43-53.

Appendix A

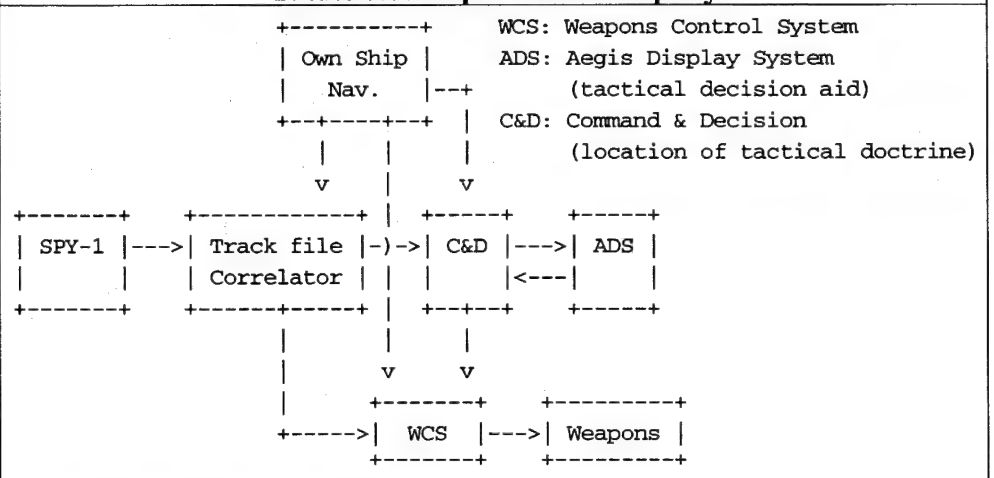
TRANSCRIPTIONS OF REAL-TIME OBSERVATIONS AND INQUIRIES

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
09:35	85	Introduce tool	Not on computer	O: Instructional style will be interactive. O: First overhead: Design model. O: Questions on clarifying design model and raising issues about formal method approach. O: Clarify finite differencing. O: Lots of questions. U: How does model handle ambiguity? T: Time needed to develop a solution. (Arrow to second overhead.) U: What type of code is produced? (Arrow pointing to "Time needed to develop a solution" item above.) O: Second overhead: Example problem, Costas Array Problem; 2 weeks to solve but this involved a false start. O: Few questions. O: Third overhead: Transportation scheduler; elaborated on KIDS concept as a knowledge-based system. O: No questions. O: Fourth overhead: Track assignment in an Air Traffic Control System; described 2 design solutions. O: No questions.

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
		Introduce tool (continued)		O: Fifth overhead: Example of a theory of sets used to illustrate how the theory is described. O: No questions. O: Sixth overhead: Example of a theory of sorting (this was the first showing [of] a specification); clarify meaning of a specification as a formal interface, distinction between domain theory and specification, notion of formality, user knowledge needed in using formal methods (i.e., requiring theorem-proving expertise). O: Lots of questions. T: Background of user of KIDS should be different; maybe training has to be greater, but code generation isn't required, design can be biased toward specific targets, clarify design model. U: Order of the options in refining the specification. T: They're a set of tools.
			Not on computer	U: How much of a learning curve for the design model used? R(T): More of a curve for US compared to European—and presumes some comprehension of mathematics.
10:26				O: Seventh overhead: Theory Development. T: Principle of how to develop theory. T: Where do rules come from? From laws. T: What laws and concepts are needed? Concepts that remain invariant and operations that preserve concepts. O: Eighth overhead: Algorithm design—refinement hierarchy; algorithm ideas can be modeled as theories. O: Ninth overhead: Variation of algorithm hierarchy from operation research. Q: What is your general reaction to this introduction of language? R(U): Totally different system; does [it] require a fundamental skill before using it? Q: What languages do users prefer?

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
		Introduce tool (continued)		R(U1): Lisp and Fortran, oriented toward vector machine; now is excited about this tool. R(U2): This could be used to get correct algorithms, favorite language is C++ with EE background. Q: How did this intro compare to typical introduction to a new language? U: Was a language introduced? O: Discussion of what the language is with this system. U: How would the tool work for maintenance? T: There could be a problem if a person was only oriented to C or for someone without a functional language background. Preferred background would be someone familiar with first order predicate calculus, sets, functional language. Q: What about an introduction to the interface commands?
11:00	5	Demo of system (Queens problem)	Load	U: Representation form—Lisp? R(T): No.
11:02			Load	U: Effect of loading extraneous theories.
11:03			Load	U: Is there a "man" facility ? R(T): Browsing mechanism only.
11:05	11	Demo of development theory	Load	K: Error after loading 2 theories—attempted to delete one, failed on first attempt.
11:06			Program window after transformation	O: Described the features of interface and representation of specification.
11:08			Program window after transformation	U: How to handle optimization for particular architecture.
11:12			Choose tactic	O: Chose global search design tactic.
11:13			Choose Rainbow	T: Pruning mechanism; control mechanism.
11:14			Choose previous	T: Recursive program.

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
11:16	7	Simplify	Prog/Spec window	U: Does simplification support understanding; can you simplify whole program?
11:23	5	Partial evaluation	Prog/Spec window	
11:28	9	Finite differencing		
11:37	2	Case analysis		
11:39	18	Compile		O: Complete derivation. U: Have you compared the results of this prototyping technique to code produced by programmers? R(T): Have not tried against programmers, only compared to standard graduate computer textbook algorithm. U: What complexity of problem has it been applied to? R(T): Scheduling. U: How long does it take a user to learn it? R(T): Not enough data points to really know.
11:57	9	Demo of derivation	Command window	O: Entered commands to find and type out derivation structure.
12:06	9	Demo of theory development (sorting theory)	Prog/Spec window	
12:15	(est)			**Break**
14:33 14:58 15:00	134 (est)	Understand problem	Not on computer	O: Group discussed the definition of engageability and doctrine zones O: Discussion of the problem covered following topics: -Presentation of the update. -AEGIS System diagram.

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
		Understand problem (continued)		 -Track file updates are no faster than 1 per second. -Form of input discussion; clarification of input data, only location or more as well. O: Discussion of generality of design including following topics: - Only 1 Aegis, 1 carrier, or more general such as 1 ship, several slaves, or 2 ships. - The reason to adopt a general design is this makes it clearer how to organize our data, e.g., to know what the inputs are. - Clarification of "ownership" of doctrine and region; e.g., ownship has weapon doctrine and engageability, other ships have slave doctrine. O: Input clarification continued. - Slave doctrine can be attached to hostiles. - ID of objects: hostile, friendly, neutral, unknown, basic. O: Clarification of input display icons (i.e., circle, square, triangle): Fixed zones—how is it given. Tight zones definition. Returning aircraft approach zones: possible output would be hostile in returning aircraft zone.

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
		Understand problem (continued)		Are zones changeable? Yes. Tight zones look like they're polygonal. O: Clarification of output: paragraph on page 8 giving objective; give region for every object not a friendly. O: Long silence.
16:15			Not on computer	O: Completed understanding of input (see above).
16:20			Not on computer	O: Completed discussion of output (see above).
16:21				**Break**
16:31		Understand problem (continued)	Not on computer	O: Started discussion of how to proceed but shifted to discussion of output. T: Talk aloud about the process—because not sure about how to proceed (very little stated about what output should be). O: Shifted to discussion of output and possibility of parallelism.
16:39			Not on computer	O: Restarted discussion of building a theory. T: What objects are there? - What attributes do they have? - Are there zones of different flavors?
16:41			Not on computer	O: Reference/grid system brought up.
16:42			Not on computer	K: In discussion of domain, the following point was made about the form of the representation: T: Computationally it will be easier to implement some types of shapes. O: Discussion of how to define shapes at a conceptual level. O: Discussion of definition of objects:

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
		Understand problem (continued)		T: What to call them—platforms/contacts/vehicles/tracks; settled on contacts. Classification of zones—different types of classifications could be used; one hierarchy possible uses ENGAGEABILITY, WEAPON, SLAVED, then TIGHT as subclasses, but types also; WEDGE, ANNULUS, WEDGED-ANNULUS.
16:53			Not on computer	O: Discussion of alternative relationships between contacts and zones: - Contacts could have a set associated with them. - Another possibility: a scenario object—packages it all together as a grammar and input could be a sentence in this grammar. - Another possibility is to parse an ASCII string.
16:57	9	Theory development	Theory development	T: Create new theory and focus on it, call it HIPER-D.
16:58				U: What does focus on it mean? R(T): Comes up in this window and can interact with it.
17:00			Load	T: Will ask us what theories to import—no idea. U: That was my question. T: Arith., sequence theory, extended, basic; just basic for now.
17:02			Edit Emacs	U: Most of those [theory imports] were things generated in previous examples. O: Editing a template.
17:03			Edit Emacs	O: Attempted to retrieve a previous example (air traffic control) to refresh memory on object classes.
17:04			Edit Emacs	O: Failure to retrieve the previous example; was located in another catalog in another directory; forgot its name.
17:05			Emacs	O: Found the example.
17:06	1	Theory development: Define object-class HIPER-D-object	Emacs	O: Defining classes using a previous example. O: Defining HIPER-D-object.

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
17:06	1	Theory development: Define object-class FLATLAND	Emacs	O: Defining FLATLAND.
17:07	4	Theory development: Define attributes	Emacs	
17:09		(FLATLAND)	Emacs	U: "Map" is a REFINE operator? R(T): Yes, all attributes are treated as far as their type as global maps, attributes link objects.
17:10			Emacs	K: Error, failure to copy text—forgot a parenthesis. T: Define zone objects later.
17:11	1	Theory development: Define object-class CONTACT	Emacs	O: Defined another class (CONTACT objects, subclass of HIPER-D objects)
17:12	7	Theory development: Define attributes (CONTACT)	Emacs	
		(coordinates)		T: How to define location, as an x and y or a pair.
17:13		(classification)	Emacs	K: Decision to define a position object later. T: Define classification attribute—friendly, hostile, etc.
17:15			Emacs	K: Comment on alternative representation of coordinates (U: May be useful to define coordinates in relative terms. We may want to return to this option. Engageability computations may be easier with relative coordinates.)
17:17		(mode) (contact zones)	Emacs	K: Decision to define "mode" later.

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
17:19	10	Theory development: Define object-class ZONE	Emacs	O: Used copy, repeated paste to set up templates once ZONE class had been defined
				O: Defining WEAPONS-DOCTRINE object-class O: Defining ENGAGEABILITY-ZONE object-class O: Defining SLAVE-DOCTRINE object-class O: Defining TIGHT-ZONE object-class U: Should a distinction be made between different tight zones?
17:20				K: Discussion of how to define attributes of zones especially the shape and subtype of the zone; resolved by discussing possible cases.
17:28			Emacs	O: Completed discussion of zone typing and described possible output grammars.
17:29	2	Theory development: Define object-class POSITION	Emacs	
17:31	13	Theory development: Define attributes (ZONE)	Emacs	K: Error, forgot about inserting attributes for zone classes, returned to attributes. K: Decision on basic zone shapes: - have a wedged annulus shape - and a polygon shape. O: Defining WD-shape.
17:36			Emacs	U: What are the two levels of access to the WEDGED-ANNULUS class? R(T): WEAPONS-DOCTRINE object, to get to its inner radius...[end of tape]
17:39			Emacs	K: Discussion of implication of different representation of zones. O: Defining SD-shape. O: Defining EZ-shape. O: Defining TZ-shape.
17:42				O: Completed attributes.
17:43				U: What is significance of stuff to right of "="? R(T): Initialization for the attribute, initial values when compiled.

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
17:44	1	Theory development: Define object-class	Emacs	O: Defining WEDGED-ANNULUS and POLYGON classes.
17:44	13	Theory development: Define attributes (WEDGED-ANNULUS)	Emacs Not on computer	O: Discussion of shape attributes. U: Angle orientation of wedge, book says weapon doctrine angle is given in degrees but fails to say from what point the degrees are measured.
17:54		(POLYGON)	Emacs	T: Will need to import sequence theory. U: Is it a fixed or arbitrary length sequence? R(T): Fixed length sequences are called tuples.
17:56				T: Have we defined everything?
17:57	1	Theory development: Define types CLASSIFIER, AIR-SEA-MODE		O: Added type (symbols) called CLASSIFIER and AIR-SEA-MODE symbols.
17:57	2 (est)	Theory laws		O: Added axioms to constrain meaning of new types. Q: What were you typing over here? R(T): Same theory in two different buffers. K: Example of usage of two representations of same artifact in order to maintain consistency in different sections.
17:58		Save solution file	Emacs	
				T: That gives us the basic ontology.
17:59	(est)			**Preceding: Day 1; Succeeding: Day 2**

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
				Preceding: Day 1; Succeeding: Day 2
09:44	1		Emacs	O: Prior day's work loaded successfully.
09:45	10	Compile	Compile/Emacs	K: Grammar error, doesn't like the underscore, dashes substituted. K: Minor change in representation due to syntax error.
09:48			Compile/Emacs	K: Error, type mismatch in x and y coordinates and map. Not important: T: Supposed to be attributes of object so should be a map from position to integer, didn't get type right.
09:53				K: Discussion of theory representation (T: Next system will distinguish between axioms and theorems, now both are included under theory-language).
09:55	5	Theory operations: Define function HIPER-D	Emacs	O: Typing in HIPER-D function. T: Trying to generate a map—CONTACT to a set of zones; will come back to input, output, and properties.
9:59			Emacs	O: Short description of map constructor.
10:00	10	Theory operations: Define function ALL-ZONES	Emacs	U: What does "for all zones" give us? R(T): Want to consider all possible zones in this scenario.
10:02				O: Tutor described the general process model, defined overall concept and then filled in details.
10:08			Emacs	Q: How did you know you already had this info? R(T): Already have this data directly from data structures, don't need separate function ALL-CONTACTS.
10:10	20	Theory operations: Define function IN-ZONE		O: Discussion of how to find intersection of CONTACT—a location with set of points—within ZONE; i.e., how to compute set of points enclosed by a polygon.
10:22			Emacs	K: T noted that this was a choice point in theory definition design.
10:24			Emacs	T: Comment on where we are in solving problem (how to define IN-ZONE predicate).

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
10:26		Theory operations: Define function IN-ZONE	Emacs	K: Error, hit return key instead of another key; was attempting to copy and paste template.
10:28		(continued)	Emacs	K: Error, hit formatting command instead of desired meta command.
10:30	51	Theory operations: Define function IN-WEDGED-ANNULUS	Emacs	K: Rewrite of higher level function. O: Second instance of revising representation or a higher level function after having started on the lower level. T: Instead of passing whole zone, only need to pass the attribute shape.
10:34			Emacs	K: Error, type mismatch noticed in writing function.
10:36			Emacs	K: Error, noticed that annulus attributes don't include origin.
10:37		Theory operations: Define function IN-WEDGED-ANNULUS	Emacs	Q: What do you mean that the optimization could be done in KIDS? R(T): An operation will be repeated several times—and this could be handled by an abstraction operation in KIDS.
10:45		(continued)	Emacs	K: Didn't remember formula to convert from cartesian to polar.
10:50				O: U1 was working independently on geometry. O: U1 offered an alternative geometric solution.
10:51				O: U2 left to find geometric function book.
11:03				O: U2 couldn't find a "particular book" that covered geometry.
11:09			Emacs	O: T described a geometric solution—and expressed limitations of reasoning from diagrams.
11:17			Emacs	O: All working alone on a geometric rationale.
11:18				T: Have top level math to figure if a point is in the wedge.
11:21	8	Finding trig function	REFINE listener	K: Error, couldn't find trig function—didn't know Lisp name of the arc function. O: U2 left to check trig function.
11:27		Save solution file	Emacs	

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
11:29	3	Theory operations: Revise function IN-WEDGED-ANNULUS	Emacs	T: Need a distance function and compute-angle function.
				O: Save theory.
11:30			Emacs	T: Completed definition of IN-WEDGED-ANNULUS.
11:31				O: U2 returned with name of arc function.
11:32	27	Theory operations: Define functions DISTANCE, COMPUTE-ANGLE	Emacs	
11:34			Emacs	K: Discussion of cases begins: T: What if distance is zero?
11:37				K: Error, logic in geometric test was incorrect. O: Found by rereading function test.
11:38		Theory operations: Define functions DISTANCE, COMPUTE-ANGLE (continued)		K: Revision of representation for specific cases. T: What to do if distance between CONTACT and center of zone is zero—realize that WEDGED-ANNULUS is a general zone shape to cover SLAVE-DOCTRINE zones as well. T: Solution for d = 0: if d = 0 then IN-ZONE, else check if within non-wedge of zone.
11:43				K: Revision of representation for specific cases. U: What do we do if there is no wedge? T: Revised IN-WEDGE logic to check if initial and final angles are not equal.
11:46				K: Discussion and mental checking for other cases, such as inner radius = 0, which covers SLAVE-DOCTRINE zones.
11:47			Evaluate	K: Error, doesn't like ellipses. K: Error, didn't put in vertical bar to separate conditionals. K: Error, dash not recognized as subtraction. K: Error, function not defined yet (IN-POLYGON). K: Error, local variable not defined (offset).
11:51				T: How do we define IN-POLYGON?

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
		Theory operations: Define functions DISTANCE, COMPUTE-ANGLE (continued)		
11:53		Save solution file	Emacs	
11:55				Q: Where are we in theory development? R(T): Defining IN-POLYGON is probably all we need and then can actually explore several implementations of all this. K: Representation issue (T: Definitions of concepts are not obvious in this domain).
11:59	(est)			**Break for lunch**
12:46	49	Theory operations: Define function IN-POLYGON	Not on computer	
12:51			Not on computer	O: U and T referred to graphics book—inside test being used to fill in.
13:00				K: Representation issue (T: Can't specify IN-POLYGON without giving a method or algorithm; specifying IN-POLYGON by giving a method is not satisfactory; would like to give a theory). O: Minutes of quiet work by all.
13:16				K: Decision to use odd-even method with line intersections; even # of intersections means point is inside polygon.
13:35	3 (est)	Theory operations: Define function FIND-LINE-INTERSECTION	Emacs	O: Coding intersection algorithm. O: Leave as stubs, fill in later.
13:38	10	Define theory	Evaluate	K: Error, type mismatch, real/integer—for angle; angle changed to real. K: Error, type mismatch, real/integer—for DISTANCE; DISTANCE changed to real.

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
13:43		Define theory (continued)	Emacs	K: Representation revision—renamed angle "a" to "contact-angle" (a = angle between reference and CONTACT). T: Should we do a floor or ceiling? U: Do you have a round function? R(T): Lisp does. O: Type mismatches led to discussion of how to convert and a test of several embedded functions such as round.
13:46			Evaluate Emacs Reader	K: Error, offset type mismatch. K: Error, didn't need to compile, just parse if correct. O: All type mismatches found in WEDGED-ANNULUS.
13:48	3	Look at spec		
13:51	7	Compile	Compile	K: Successful on first try. K: Error, some type of abstraction failed (abstract out a new specification).
13:58	35	Algorithm design	Tactic: Divide and Conquer	
13:59			Tactic	
14:03				K: (T: Bug in the program scheme giving the wrong variable).
14:03			Backup to disk	T: Backing up to Divide and Conquer to see where flaw originated.
14:10				K: (T: Bug in the inference engine— may take long time to find).
14:11			Restore	
14:13			Abstract Tactic	O: Exploring design options to find zones.
14:17			Tactic: Divide and Conquer	K: Error in Divide and Conquer tactic when applied to zone (same problem again—inference failure, typically because of missing laws). T: Discontinue the derivation.
14:27				T: Examining an internal theory structure—looks like it was constructed wrong.

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
14:30				**Break**
14:39		Algorithm design (continued)		O: Checked previous programs.
14:42	81	Theory operations: Define function	Emacs	O: Continued to fill out polygon function.
15:25		IN-POLYGON	Not on computer	K: Options discussed for finding IN-POLYGON. U: Assume convex polygons. U: Do a polygon fill and check for point.
15:56			Emacs	K: Proliferation of cases usually means that there isn't a good theory defined.
16:03	13	Theory operations: Define function FIND-SEGMENT-INTERSECTION	Emacs	K: Error, noticed type definition for output type wrong (change from function that evaluates line intersections to FIND-SEGMENT-INTERSECTION, which returns a segment).
16:11				O: Completion of segment intersection.
16:16	20	Check IN-POLYGON		O: Began to implement a test capability.
16:20			2 Emacs windows	O: Task switched to double buffer.
16:23			2 Emacs windows	T: Need to consider sequences is driven by the special case.
16:36	39	Theory operations: Define function FIND-SEGMENT-INTERSECTION	Emacs	T: Realized that intersection checking had to be restricted to segments that could overlap.
16:39			Emacs	K: Error, in keyboard entry [not verbalized on tape].
16:55			Emacs	T: Which is varying, x or y?
17:01			Emacs	T: Recalled what the "dangling else" was—the normal case.
17:06			Read/compile	K: Error, type mismatches.
17:08			Read/compile	K: Successful compile —can count intersections.
17:10		Save solution file	Emacs	

Time	Dur	Task	Appl./Operation	K: key incident O: observation Q: observer query R: response to query U: user comment/query T: tutor/tool-expert comment/query
17:11		Theory operations: Define function FIND-SEGMENT-INTERSECTION (continued)		Q: Clarification of parallelism of IN-POLYGON.
17:13			Emacs	K: Compilation complete. Q: Clarify what is meant by uncertainty about accurate specification.
17:15	1	Theory miscellaneous forms	Emacs	O: Began to define function to check COUNT-INTERSECTIONS of some polygon.
17:16	10	Theory operations: Define function MAKE-TEST-ZONE	Emacs double buffer	K: Decided to make a function to test zones rather than a miscellaneous form.
17:18			Emacs	K: Error, lost test—retrieved.
17:19				T: Would like to have a grammar that could be used to parse an input string for data input. Have to use a series of assignments instead.
17:23			Evaluate	K: Possible error in attempting to call function because some of the structure hadn't been named.
17:26	2	Theory operations: Define function MAKE-TEST-CONTACT	Emacs double buffer	O: Began to make function to define a contact object.
17:28	33	Test contact IN-POLYGON	Emacs double buffer	K: Error, put on trace mechanisms for error type mismatch for functions (accessing data incorrectly—IN-POLYGON expecting a polygon, not a TIGHT-ZONE).
17:32			Emacs	K: Error, divide by zero (slope of line = 0; if line is horizontal, simplify the test for finding intersection).
18:01		Save solution file	Emacs	

Appendix B
LAST SOLUTION FILE

```
%%% -*- Mode: RE; Package: RE; Base: 10.; Syntax: Refine  -*-

!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'REGROUP)

THEORY HIPER-D-4

%----------------------------------------------------------------------
THEORY-IMPORTS {}

%----------------------------------------------------------------------
THEORY-TYPE-PARAMETERS {}

%----------------------------------------------------------------------
THEORY-TYPES

type CLASSIFIER = symbol
type AIR-SEA-MODE = symbol

var HIPER-D-object : OBJECT-CLASS subtype-of USER-OBJECT

var FLATLAND : OBJECT-CLASS subtype-of HIPER-D-object
var contacts : map(flatland, set(CONTACT)) = {| |}
var flatland-zones : map(flatland, set(ZONE)) = {||}

var CONTACT : OBJECT-CLASS subtype-of HIPER-D-object
var coordinates : map(CONTACT, POSITION) = {| |}
var classification  : map(CONTACT, CLASSIFIER) = {| |}
var mode : map(CONTACT, AIR-SEA-MODE) = {| |}
var contact-zones : map(CONTACT, set(ZONE)) = {| |}

var ZONE : OBJECT-CLASS subtype-of HIPER-D-object
var WEAPONS-DOCTRINE : OBJECT-CLASS subtype-of ZONE
var SLAVE-DOCTRINE : OBJECT-CLASS subtype-of ZONE
var ENGAGEABILITY-ZONE : OBJECT-CLASS subtype-of ZONE
var TIGHT-ZONE : OBJECT-CLASS subtype-of ZONE

var WD-shape : map(WEAPONS-DOCTRINE, WEDGED-ANNULUS) = {||}
var SD-shape : map(SLAVE-DOCTRINE, WEDGED-ANNULUS) = {||}
var EZ-shape : map(ENGAGEABILITY-ZONE, WEDGED-ANNULUS) = {||}
var TZ-shape : map(TIGHT-ZONE, POLYGON) = {||}
```

```
var WEDGED-ANNULUS : OBJECT-CLASS subtype-of HIPER-D-object
var inner-radius : map(WEDGED-ANNULUS, integer) = {||}
var outer-radius : map(WEDGED-ANNULUS, integer) = {||}
var initial-angle : map(WEDGED-ANNULUS, integer) = {||}
var final-angle : map(WEDGED-ANNULUS, integer) = {||}
var origin : map(WEDGED-ANNULUS, POSITION) = {||}

var POLYGON : OBJECT-CLASS subtype-of HIPER-D-object
var vertices : map(POLYGON, seq(POSITION)) = {||}

var POSITION : OBJECT-CLASS subtype-of HIPER-D-object
var x-coord : map(POSITION,integer) = {| |}
var y-coord : map(POSITION,integer) = {| |}

type angle = integer

%-----------------------------------------------------------------------
THEORY-OPERATIONS

function HIPER-D
  ( fo : FLATLAND )
  returns ( contact-map : map(CONTACT, set(ZONE))
       | contact-map = {| c -> { z | (z:ZONE) z in all-zones(fo) & in-zone( c,z)}
                | (c:CONTACT) c in contacts(fo) |})

function HIPER-D-2 (CNTKS-7: set(CONTACT), ZNS-9: set(ZONE))
 returns
  (Z-192: map(CONTACT, set(ZONE))
    | Z-192
      = {| C -> {Z | (Z: ZONE)
              Z in ZNS-9 & IN-ZONE(C, Z)}
        | (C: CONTACT) C in CNTKS-7 |})
 = if CNTKS-7 = { } then {| |}
   elseif CNTKS-7 less! arb(CNTKS-7) = { }
      then {| C -> {Z | (Z: ZONE)
                Z in ZNS-9 & IN-ZONE(C, Z)}
           | (C: CONTACT) C in CNTKS-7 |}
   else let (Y-OP-3
          : tuple
            (set(CONTACT), set(ZONE), set(CONTACT), set(ZONE))
          = HIPER-D-2-DECOMPOSE-USING-UNION-DESTRUCTOR
            (CNTKS-7, ZNS-9))
       HIPER-D-2(Y-OP-3.1, Y-OP-3.2)
        +* HIPER-D-2(Y-OP-3.3, Y-OP-3.4)

function ALL-ZONES
  ( fo : FLATLAND ) : set(ZONE)
  = flatland-zones(fo) union reduce(union, image(contact-zones, contacts(fo)))
```

```
function IN-ZONE
  (c : CONTACT, z : ZONE) : boolean
  = (if WEAPONS-DOCTRINE(z) then in-wedged-annulus(c,WD-shape(z))
    elseif SLAVE-DOCTRINE(z) then in-wedged-annulus(c,SD-shape(z))
    elseif ENGAGEABILITY-ZONE(z) then in-wedged-annulus(c,EZ-shape(z))
    elseif TIGHT-ZONE(z) then in-polygon(c,TZ-shape(z))
    else undefined)

function IN-WEDGED-ANNULUS
  (c : CONTACT, z : WEDGED-ANNULUS ) : boolean
  = (let (p : POSITION = coordinates(c))
    <x-coord(p), y-coord(p)>
    in
    { <x,y> | (x:integer, y:integer, d : real, contact-angle : angle, offset:  integer)
          d = distance(x-coord(origin(z)), y-coord(origin(z)),x,y)
          & inner-radius(z) <= d & d <= outer-radius(z)
          & (d > 0.0
          & initial-angle(z) ~= final-angle(z)
          => contact-angle = compute-angle(x-coord(origin(z)), y-coord( origin(z)),x,y, d)
            & offset = (contact-angle - initial-angle(z)) mod 360
              & offset <= (final-angle(z) - initial-angle(z)) mod 360)
     })

function DISTANCE
  (x1 : integer, y1 : integer, x2 : integer, y2 : integer) : real
  = sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1))

% note: we have rounded, so this is approximate
function COMPUTE-ANGLE
  (x1 : integer, y1 : integer, x2 : integer, y2 : integer, d : real
   | d = distance(x1,y1,x2,y2) & d > 0.0) : integer
  = round(acos( (x2 - x1) / d))

function FIND-SEGMENT-INTERSECTION
  (x1 : integer, y1 : integer, x2 : integer, y2 : integer,
   u1 : integer, v1 : integer, u2 : integer, v2 : integer)
  : tuple(real, real,real,real)
  = (let (slope-xy : real = (x2 - x1)/(y2 - y1),
       slope-uv : real = (u2 - u1)/(v2 - v1))
    let (intercept-xy : real = y1 - slope-xy * x1,
      intercept-uv : real = v1 - slope-uv * u1)
    if slope-xy = slope-uv
    then (if intercept-xy ~= intercept-uv
          then undefined
          else check-coincident-lines(x1, y1, x2, y2,u1, v1, u2, v2))
    else (let (x : real = (intercept-uv - intercept-xy)/(slope-uv - slope-xy))
       let (y : real = slope-xy * x + intercept-xy)
       (if x1 <= x2
        then (if x1 <= x & x <= x2
              then <x,y,x,y>
             else undefined)
```

```
        else (if x2 <= x & x <= x1
            then <x,y,x,y>
           else undefined))))

function CHECK-COINCIDENT-LINES
  (x1 : integer, y1 : integer, x2 : integer, y2 : integer,
   u1 : integer, v1 : integer, u2 : integer, v2 : integer)
  : tuple(real, real,real,real)
  = undefined

function COUNT-INTERSECTIONS
  (cx : integer, cy : integer, outx : integer, outy : integer,
   z-vertices : seq(POSITION), cnt : integer) : integer
  = (if size(z-vertices) <= 1
     then cnt
     else (let (Intersect-interval : tuple(real, real,real,real)
          = find-segment-intersection(cx,cy,outx,outy,
                        x-coord(first(z-vertices)),
                        y-coord(first(z-vertices)),
                        x-coord(second(z-vertices)),
                        y-coord(second(z-vertices))))
       if defined?(Intersect-interval)
       then (if Intersect-interval.1 = Intersect-interval.3
            & Intersect-interval.2 = Intersect-interval.4
          then % single point intersection
             (if Intersect-interval.1 = integer-to-real(x-coord(first( z-vertices)))
             & Intersect-interval.2 = integer-to-real(y-coord(first( z-vertices)))
              then undefined  %elaborate later
             elseif Intersect-interval.1 = integer-to-real(x-coord(se cond(z-vertices)))
                & Intersect-interval.2 = integer-to-real(y-coord(se cond(z-vertices)))
              then (if y-coord(first(z-vertices)) <= cy
                 then (if y-coord(second(z-vertices)) <= cy
                    then COUNT-INTERSECTIONS(cx,cy,outx,outy,
                              rest(rest(z-verti ces)), cnt)
                    else COUNT-INTERSECTIONS(cx,cy,outx,outy,
                              rest(rest(z-verti ces)), cnt + 1))
                 else (if y-coord(second(z-vertices)) <= cy
                    then COUNT-INTERSECTIONS(cx,cy,outx,outy,
                              rest(rest(z-verti ces)), cnt + 1)
                    else COUNT-INTERSECTIONS(cx,cy,outx,outy,
                              rest(rest(z-verti ces)), cnt)))
          else %normal case
            COUNT-INTERSECTIONS(cx,cy,outx,outy, rest(z-vertices),  cnt + 1))
        else % coiincident lines
           undefined)
       else % no intersection
       COUNT-INTERSECTIONS(cx,cy,outx,outy, rest(z-vertices), cnt)))

function IN-POLYGON
  (c : CONTACT, z : POLYGON ) : boolean
  = (let (p : POSITION = coordinates(c),
```

```
      z-vertices : seq(POSITION) = vertices(z))
    let (cx = x-coord(p), cy = y-coord(p),
      outx = -1, outy = y-coord(p))
    if count-intersections(cx, cy, outx, outy,
                z-vertices ++ [first(z-vertices)], 0) = 1 mod 2
    then true
    else false
    )

function MAKE-TEST-ZONE ()
 = (let (z : ZONE = make-object('TIGHT-ZONE),
      poly : POLYGON = make-object('POLYGON),
      p1 : POSITION = make-object('POSITION),
      p2 : POSITION = make-object('POSITION),
      p3 : POSITION = make-object('POSITION),
      p4 : POSITION = make-object('POSITION),
      p5 : POSITION = make-object('POSITION),
      p6 : POSITION = make-object('POSITION)
      )
    x-coord(p1) <- 0;   y-coord(p1) <- 25;
    x-coord(p2) <- 118; y-coord(p2) <- 62;
    x-coord(p3) <- 259; y-coord(p3) <- 25;
    x-coord(p4) <- 259; y-coord(p4) <- 5;
    x-coord(p5) <- 118; y-coord(p5) <- 32;
    x-coord(p6) <- 0;   y-coord(p6) <- 5;
    vertices(poly) <- [p1,p2,p3,p4,p5,p6];
    TZ-shape(z) <- poly;
    poly)

function MAKE-TEST-CONTACT ()
 = (let (c : CONTACT = make-object('CONTACT),
      p1 : POSITION = make-object('POSITION))
    x-coord(p1) <- 258;   y-coord(p1) <- 183;
    coordinates(c) <- p1;
    c)

%-----------------------------------------------------------------------
THEORY-LAWS

assert def-of-CLASSIFIER
 fa(c : CLASSIFIER) c in {'friendly, 'hostile, 'own, 'unknown}

assert def-of-AIR-SEA-MODE
 fa(asm : AIR-SEA-MODE) asm in {'AIR, 'SEA}

%-----------------------------------------------------------------------
THEORY-RULES

%-----------------------------------------------------------------------
THEORY-MISC-LAWS
```

```
%-----------------------------------------------------------------------
THEORY-MISC-DEFS

%-----------------------------------------------------------------------
THEORY-MISC-RULES

%-----------------------------------------------------------------------
THEORY-MISC-FORMS

%-----------------------------------------------------------------------
end-theory
```

Appendix C
QUESTIONNAIRE

The following questions address the software development model in the tool. These questions address several functions in the model. Each function is indicated with italics. For each function, please give a general assessment, as well as a specific assessment of the best features and any recommendations you would have for changes.

Scale	0	1	2	3	4	5
Rating	Poor	Fair	Average	Above Average	Good	Excellent

Q1. In general, how well does the tool support the development of a theory of the domain?

What are the tool's best features for this function?

What changes or additions would you suggest for this function?

Q2. As a part of developing the domain theory and specification, how well does the tool support *specifying functional constraints on input/output behavior*?

What are the tool's best features for this function?

What changes or additions would you suggest for this function?

Q3. As a part of developing the domain theory and specification, how well does the tool support the *generation of rules and derivation of laws*?

What are the tool's best features for this function?

What changes or additions would you suggest for this function ?

Q4. In general, how well does the tool support converting a specification to code?

What are the tool's best features for this function?

What changes or additions would you suggest for this function ?

Q5. As part of converting a specification to code, how well does the tool support *algorithm design*?

What are the tool's best features for this function?

What changes or additions would you suggest for this function ?

Q6. As part of converting a specification to code, how well does the tool support *algorithm simplification*?

What are the tool's best features for this function?

What changes or additions would you suggest for this function ?

Q7. As part of converting a specification to code, how well does the tool support *partial evaluation*?

What are the tool's best features for this function?

What changes or additions would you suggest for this function ?

Q8. As part of converting a specification to code, how well does the tool support *refinement of data types*?

What are the tool's best features for this function?

What changes or additions would you suggest for this function ?

The next few questions address how the tool works in supporting software design.

Q9. How well does the tool support exploration of the design space (i.e., exploration of alternative design solutions)?

Q10. How well does the tool support the *generation* of alternative designs?

Q11. How well does the tool support the *evaluation* of alternative designs?

Q12. What are the strengths of the tool?

Q13. What are the weaknesses of the tool?

Q14. In general, what additional functionality would you like to see?

Q15. What changes would you suggest to the interface?

Q16. Is the symbology appropriate in the interface?

Q17. Are the graphics adequate?

Q18. Is the response time of the tool to user commands adequate?

Q19. Is the software model implemented in this tool consistent with your concept of the software development process?

Q20. Is the problem that the tool was used on sufficiently complex to assess the utility of the tool for designing software for Navy systems?

The following features have been found important in the usability of the interface. Please comment on the extent to which these features are present in the tool, and elaborate on your assessment.

Feature	Present	Elaboration
Simple and natural dialogue		
Speaks the user's language		
Minimizes user memory load		
Consistency		
Provides feedback		
Provides clearly marked exits		
Provides shortcuts		
Provides good error messages		
Designed to prevent errors		
Tolerates errors		
Provides for error recovery		
Minimizes system response time		